User-Oriented Dynamic MEC Application Deployment in Edge Cloud Network

Yinan Guo[1], Yanzhao Hou[2(✉)], Jiaxiang Geng[1], Hao Chen[3], and Whai-En Chen[4]

[1] Beijing University of Posts and Telecommunications, Beijing, China
[2] Beijing University of Posts and Telecommunications Shenzhen Institute, Shenzhen, China
houyanzhao@bupt.edu.cn
[3] Department of Broadband Communication, Peng Cheng Laboratory, Shenzhen, China
[4] Department of Computer Science and Information Engineering, National Ilan University, Yilan, Taiwan

Abstract. Multi-Access Edge Computing (MEC) have become the core technologies to meet users' needs for 5G and beyond wireless networks. MEC applications can be flexibly created and placed at the network edge through virtual network functions (VNFs) to provide users with specific services with lower latency. In this paper, we consider a multi-user dynamic MEC network, where user trajectories follow Lévy walks, and each user has a set of requested target MEC applications. Our goal is to obtain an online deployment algorithm that is able to serve dynamic user requests within a tolerable latency while balancing the computational load among Mobile Edge Platforms (MEPs) as much as possible. This requires real-time processing of intractable NP-hard optimization problems. To tackle this problem, we propose an online deployment framework for MEC applications based on deep reinforcement learning, whose policy-based features adapt to the characteristics of the large action space in the problem. The framework learns binary deployment decisions from experience without solving NP-hard optimization problems, which greatly reduces computational complexity. Through simulations, we demonstrate the ability of our scheme to balance the computational load among the available MEPs and to satisfy the dynamic service requests of users.

Keywords: MEC application placement · Multi-access Edge Computing (MEC) · Reinforcement learning

1 Introduction

Ultra Reliable Low Latency Communications (URLLC), one of the typical application scenarios of 5G, requires high reliability and ultra-low latency access (about 1ms in the radio access network (RAN)) [1,2]. Traditionally, centralized cloud-based applications cannot meet the low-latency requirements of some

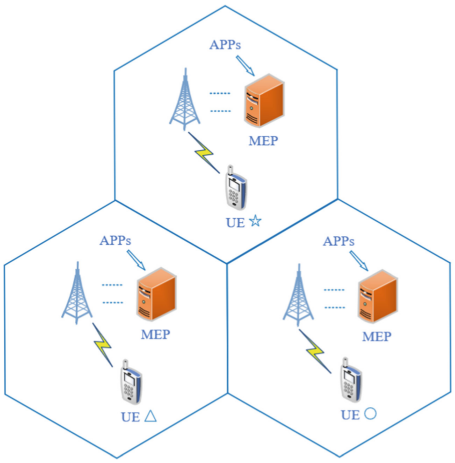
K. Rabie et al. (Eds.): IoTaaS 2022, LNICST 506, pp. 82–93, 2023.
https://doi.org/10.1007/978-3-031-37139-4_8

real-time applications due to their long-distance communication with user equipment. In this case, multi-access edge computing (MEC) can deploy applications or some of their components at the edge in the form of virtual network functions (VNFs) to maintain low latency for key URLLC services [3]. Based on the standard ETSI MEC model [4], MEC applications are deployed on the MEC platform (MEP) in the form of virtual machines or containers. According to the operator's deployment strategy, one MEP can cover a group of gNodeBs. We consider MEC applications as a whole and can only be deployed on a single MEP. When deploying MEC applications at the edge, the MEC orchestrator (MEO) must select the best MEP to meet the application latency and required resource capacity (CPU, storage, etc.). However, in actual deployment, some MEPs may not be able to carry some MEC applications, mainly due to the following three difficulties: (i) Unlike the centralized cloud with super-large capacity located in the data center, the computing power of a single MEP cannot meet the requirements of some MEC applications for computing or storage resources, and the number of MEC applications that it can carry is also limited; (ii)The unpredictable time-varying wireless channel conditions brought about by the randomness of user distribution lead to uncertain transmission delays between users and MEPs, and it is difficult to guarantee the tolerance delay of MEC applications required by users; (iii) Considering the diverse dynamic service requests of different users, it is difficult to quickly meet the needs of all users for the required MEC applications. Therefore, the challenging question raised by the above difficulties is where to deploy MEC applications in order to better meet their application requirements in terms of computing and storage resources and latency, while meeting the user's dynamic specific service needs.

Only a few works address the placement of MEC applications in edge clouds. For example, a tabu search-based deployment algorithm for MEC applications in federated MEPs is proposed in [5], which uses short-term memory to iteratively find optimal solutions from a neighborhood solution space. Both [6,7] have made meaningful work on the deployment of MEC applications, and [6] proposed a method to schedule the optimal location of MEC applications based on temporal network-wide delay fluctuations using optimal stopping theory, [7] proposed a genetic-based heuristic algorithm to solve the optimal placement problem for MEC applications. However, these works either do not consider the changing network state caused by the user's movement pattern, or require a large number of iterations to reach a satisfactory local optimum, and crucially, do not consider the needs of different users for specific MEC applications

Our work is inspired by the advantages of deep reinforcement learning in dealing with reinforcement learning problems with large state and action spaces. At present, there are some works that use deep reinforcement learning to study the task offloading decision problem in MEC networks. MEC application deployment decision problem is similar to it, both problems are learned from training data samples, and finally generated Integer decisions and assess their quality, so we can learn from and reference. [8,9] studied task offloading strategies in MEC networks based on DQN. [10,11] studied offloading and resource allocation based on dual DQN and duel DQN, respectively. [12] proposed a mapping

method to improve the slow convergence of DQN in a large action space, but its algorithm is more likely to fall into local optimum in some scenarios. In general, these value-based reinforcement learning algorithms are not suitable for handling problems with high-dimensional action spaces, and it is difficult to obtain good convergence in our problem.



MEP to MEP	High-speed Xn interface connection
UE to MEP	Wireless channel and the UE trajectory follows Lévy walks
MEC Applications	A set of MEC applications with different computing resources and latency requirements will be deployed on MEP
Target MEC Applications List	Each user has a different target applications request. E.g, △ ☆ ○ represents a different list

Fig. 1. An example of the considered MEC network, Where the MEP is attached to the gNodeB, all User Equipments (UEs) are dynamically mobile and have requests for specific MEC applications.

In this paper, we consider a multi-user MEC network, as shown in Fig. 1, each MEP can deploy MEC applications within its available resources, and the MEPs are connected through a high-speed Xn interface. In particular, there are multiple users randomly distributed in each cell, and each user has a specific service request. Our goal is to optimize deployment decisions for MEC applications. To this end, we propose a deep reinforcement learning-based online MEC application deployment framework to balance the computational load among the available MEPs, while ensuring that the user access latency of MEC applications satisfies constraints, as well as satisfying user-specific service requests. Our contributions are summarized as follows:

- A user-oriented MEC application deployment problem was studied that considers user-specific service requests and latency uncertainty caused by user mobility, with the goal of balancing the computational load among MEPs.
- The problem is shown to be NP-hard, and the search space for its solution grows exponentially. The proposed algorithm learns from past deployment experience under user distribution and user service request conditions, and automatically improves its action generation policy, so the computational complexity does not explode with network size.

– Different from many existing value-based reinforcement learning algorithms, our proposed algorithm is policy-based, and it also has a good convergence effect when the number of applications to be deployed is large and the action space is large, avoiding the curse of dimensionality.
– The performance of the proposed algorithm is demonstrated by simulations, showing that our scheme can balance the load among MEPs under various user distributions and different service requests, while ensuring that the requirements of MEC applications are met.

The rest of this article is organized as follows. Section 2 introduces the system model and formulates the optimization problem. Section 3 introduces the detailed design of the proposed algorithm. The obtained experimental results are discussed in Sect. 4. Finally, the paper is concluded in Sect. 5.

2 System Model

As shown in Fig 1, there is one MEP attached to the gNodeB in each cell, and there are E MEPs in total, denoted as $\mathcal{E} = \{1, 2, ..., E\}$, the available computing resources of the ith MEP is denoted as R_i, and their values vary in order to simulate real scenarios. There are M MEC applications to be deployed, denoted as $\mathcal{M} = \{1, 2, ..., M\}$, the maximum tolerable delay of the jth MEC application is denoted as l_j, and the required computing resources are denoted as r_j. There are N user equipments (UEs) in the network, denoted as $\mathcal{N} = \{1, 2, ..., N\}$. Each UE has a set of target MEC applications, and the target MEC application list of the kth UE is denoted as v_k. There is wireless channel transmission between UE and MEP, and high-speed Xn interface connection between MEPs, which can transmit service data of MEC applications. In this paper, we will make MEC application deployment decisions based on the user's dynamic channel conditions, as well as their list of different target MEC applications.

2.1 User Mobility

Due to the mobility of users, the deployment of MEC applications faces challenges. On the one hand, it is necessary to ensure that the target applications requested by users can arrive within the maximum tolerable delay. On the other hand, the capability difference between MEPs should also be considered, so as to avoid a system crash caused by the overload of a certain MEP as much as possible. Lévy walks [13] are a widely used way of simulating human flow trajectories. In this paper, the action trajectories of N users in the MEC network conform to Lévy walks. The user's initial position is randomly distributed in each cell, and to constrain the step size to the simulation region, we use a Lévy stable distribution to generate user trajectories. The coordinate set of the user at time t in the network area is recorded as $\mathbf{c}_t$.

2.2 Delay Model

In this paper, we consider a typical Orthogonal Frequency Division Multiplexing(OFDM) network with a frequency reuse factor of 1. There are K subchannels in each cell, the subchannel bandwidth is denoted as B, and the subchannels used by users are randomly allocated. The transmit power allocated by the base station to a single user is P, the white Gaussian noise power is denoted as σ^2, and the path loss model refers to the typical empirical formula of the UMi scene in 3GPP TR 38.901 [14], denoted as h_k, which can be expressed as

$$h_k = 32.4 + 21 \log_{10}(d_k) + 20 \log_{10}(f_c) \tag{1}$$

where d_k is the distance from the gNodeB to the user k, and f_c is the center carrier frequency.

The scenario considered in this paper is that the MEC application provides services for users, so the interference mainly comes from the downlink transmission power of the base station in the cell where the user sharing the subchannel resides. The distance between the base station generating interference and the user k can be denoted as d_{ik}, so the interference power can be expressed as

$$N_k = \sum_i h_{ik} P \tag{2}$$

Based on the above definition, the downlink transmission rate of the $k-th$ user can be expressed as

$$R_k = B \log_2 \left(1 + \frac{h_k P}{N_k + \sigma^2}\right) \tag{3}$$

Denote the size of the data packet delivered when the MEC application provides services as D_j, assuming that service data can be transmitted between MEPs through the fixed-rate Xn interface, the transmission rate is denoted as R^*, and the processing delay of one forwarding is denoted as t^*, then the delay of user k when it obtains its target MEC application j deployed in MEPi can be expressed as

$$T_{ik} = \begin{cases} D/R_k & \text{if application } j \text{ in } k\text{'s cell} \\ D/R_k + D/R^* + \alpha t^* & \text{otherwise} \end{cases} \tag{4}$$

where α is the forwarding times.

2.3 Problem Formulation

The goal of this paper is to balance the computing load among MEPs as much as possible, and the balance degree index can be formulated as follows:

$$Q(\mathbf{c}, \mathbf{x}, \mathbf{v}) \triangleq \sum_{i=1}^{i=E-1} \sum_{i'=i+1}^{i'=E} \left(\left|\frac{\sum_{j=1}^{j=M} x(i,j) r_j}{R_i} - \frac{\sum_{j=1}^{j=M} x(i',j) r_j}{R_{i'}}\right|\right) \tag{5}$$

where $\mathbf{c} = \{c_k \mid k \in \mathcal{N}\}$, $\mathbf{v} = \{v_k \mid k \in \mathcal{N}\}$, and $\mathbf{x}$ represents the binary deployment decision.

In order to find optimal or satisfactory sub-optimal application deployment decisions, satisfy the requests of different users for MEC applications, and ensure the delay and resource constraints required by these MEC applications, based on the previous definition, the optimization problem can be formulated as follows :

$$\min Q(\mathbf{c}, \mathbf{x}, \mathbf{v}) \tag{6}$$

s.t.

$$\begin{aligned} C1 &: \sum_{i=1}^{i=E} X(i,j) = 1, \forall j \in \mathcal{M} \\ C2 &: \sum_{i=1}^{i=E} \sum_{j=1}^{j=M} X(i,j) r_j \leq R_i \\ C3 &: X(i,j) T_{ik} \leq l_j, \quad \forall j \in v_k, k \in N, i \in E \end{aligned} \tag{7}$$

The problem (6) is to minimize the sum of load percentage differences between MEPs. $X(i,j)$ is the deployment decision matrix of MEC applications, where $x(i,j) = 1$ if application j is deployed on the MEP i. $C1$ guarantees that a MEC application can only be deployed on one MEP. $C2$ guarantees that the resources occupied by MEC applications on each MEP do not exceed the available resources. $C3$ guarantees that the user's delay in using the service of the target MEC application is less than the maximum tolerable delay of the application. It can be seen that problem (6) is an NP-hard problem.

The main difficulty in solving problem (6) lies in the deployment decision. Traditional optimization algorithms need to iteratively adjust deployment decisions to achieve optimality, which is infeasible when the number of users and MEC applications is large. In order to solve the complexity problem, we propose a novel online deployment algorithm based on deep reinforcement learning, whose policy-based characteristics can be well applied to large action spaces and meet the needs of users.

3 The Proposed Algorithm

Our goal is to design a deployment strategy function π, which can quickly generate optimal deployment actions. For example, when there are 4 MEPs, the deployment decision of each MEC application can be represented by 2 bits, so the deployment decision matrix X can be Reshape into a decision vector $\mathbf{x} \in \{0,1\}^{2M}$ of length $2M$. As users move within the network area, we take user coordinates $\mathbf{c}$ every time period T as input to the function, which then gives deployment decisions. This strategy can be expressed as

$$\pi : \mathbf{c} \rightarrow \mathbf{x} \tag{8}$$

The proposed algorithm gradually learns such a policy function π from experience.

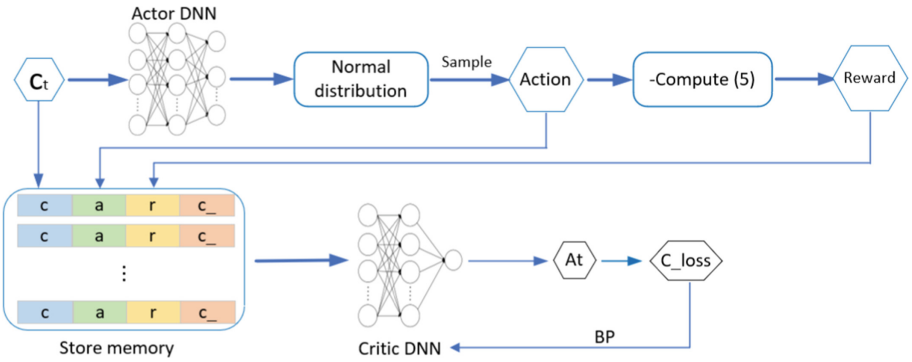


Fig. 2. Partial schematic diagram of the proposed algorithm without policy updates.

3.1 Deployment Action Generation

Part of the structure of the proposed algorithm is shown in Fig. 2. It consists of two alternating phases: deployment action generation and policy update. First, the deployment action generation is introduced. The user coordinate $\mathbf{c}_t$ at t time is input into the actor network, the network will construct a normal distribution, perform multiple sampling and output an action, and quantify it to get the deployment vector $\mathbf{x}$. Taking the existence of 4 MEPs as an example, the original sampling output of the network is the continuous value of **action** $\in (0,1)^{2M}$, we quantify it to get the required deployment decision $\mathbf{x} \in \{0,1\}^{2M}$, the quantization rule is shown in Eq. (9)

$$x_{t,j} = \begin{cases} 1 & \text{action }_{t,j} > 0.5 \\ 0 & \text{action }_{t,j} \leq 0.5 \end{cases} \tag{9}$$

for $j = 1, ..., 2M$

Then use $\mathbf{c}, \mathbf{x}, \mathbf{v}$ as a parameter to calculate formula (5), and since reinforcement learning uses gradient ascent to maximize rewards and, we take its negative value as the reward for the current action. When the constraints in (7) are not met, we will punish the action and reduce the reward value. Store this set of $[\mathbf{c}_t, \mathbf{a}_t, r, \mathbf{c}_{t+1}]$ in store memory, and then loop this process multiple times until the number is at least reach a batch.

3.2 Policy Update

Our scheme is a deep reinforcement learning algorithm based on PPO (Proximal Policy Optimization) [15], and the data stored in store memory will be used to update the parameters of Deep Neural Network (DNN). Overall, our goal is to make the deployment actions output by the network have an advantage, for which the proposed algorithm uses an actor-critic architecture. The At calculated by the critic network output is called the advantage function, which can be

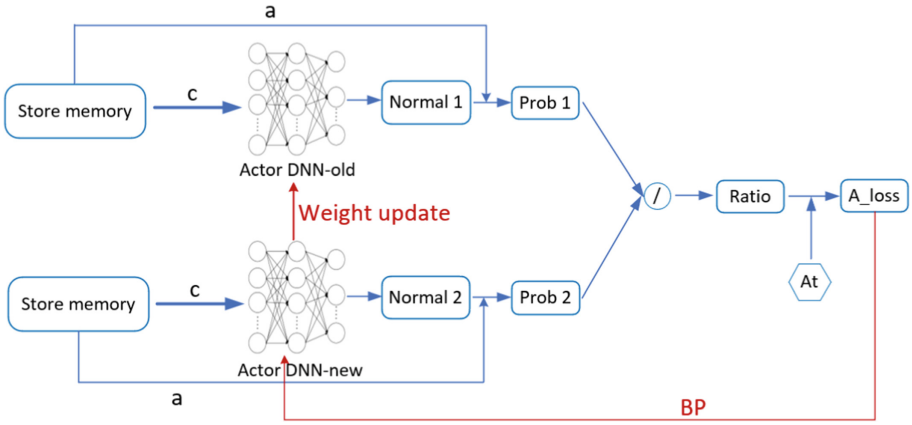


Fig. 3. Schematic diagram of policy update of proposed algorithm.

used to evaluate the advantage of the actor network output action. It should be pointed out that the At here is the cumulative conversion advantage of the entire experience pool. As shown in Fig. 2, the critical network output state value is used to estimate the advantage function value At, which uses MSE loss as the loss function, and updates the network parameters through backpropagation to achieve a more accurate estimate.

Then, as shown in Fig. 3, the actor network starts to update. Through the $[\mathbf{c}, \mathbf{a}]$ stored in memory and the At obtained by using the critic network, use the following loss function to update the parameters:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right] \tag{10}$$

where $r_t(\theta)$ represents the state-action probability ratio between the new policy and the old policy, and ϵ is a hyperparameter that controls the distance between the old and new policies. The actor DNN uses this backpropagation to update its own network parameters, thereby maximizing the advantage, that is, getting the maximum reward. We provide pseudocode for our scheme in Algorithm 1.

4 Simulation Results

4.1 Simulation Setup

In this section, we use simulations to evaluate the performance of the proposed algorithm. We simulated a scenario with 4 MEPs, the Gaussian white noise power is -114 dBm, and the downlink transmit power is 29 dBm. Each user occupies 5 Resource Blocks (RBs), plus the protection bandwidth, a total of 1 MHz. Then, the occupied resources r_j and the maximum tolerable delay l_j of each MEC application are randomly generated. We use Pytorch [16] to implement the proposed algorithm in Python, and set the policy update interval delta = 256, the

Algorithm 1. An Online Algorithm for Solving MEC Application Deployment Decision Problem

Require: $\mathbf{c}_t, h_k, \mathcal{L}_k, R_i, l_j, r_j, D_j, R^*$, penalty parameter$\beta$
Ensure: MEC applications deployment decision vector $\mathbf{x}$

```
The random parameter θ_1 initializes the actor DNN, the random parameter ω_1 initializes the critic DNN, and clears the storage.
Set the number of iterations P, the training interval δ, and initialize reward_0.
for t = 1, 2, ..., P do
    The actor DNN outputs deployment decisions and quantifies, getting x_t = f_{θ_t}(x_t)
    if Constraint (7) is satisfied then
        Calculate reward_t = −Q(c_t, x_t)
    else
        reward_t = reward_{t−1} − β
    end if
    Save [c_t, x_t, reward_t, c_{t+1}] to memory
    if t%δ = 0 then
        Input memory into critic DNN
        Calculate A_t and use MSE loss to update parameter ω_t
        Input memory into actor DNN-old and actor DNN-new
        Use function (10) as the loss function to update the parameter θ_t
        Clear memory
    end if
end for
```

policy evaluation interval 5000, the learning rate of both actor and critic are 0.0003, and the penalty parameter $\beta = 0.05$.

To verify the performance of our scheme, we compare it with the DROO algorithm and the unbalanced load algorithm, where the unbalanced load algorithm only ensures that the delay and resource constraints of MEC applications are met, and does not pursue load balance.

4.2 Simulation Results

In Fig. 4(a), we plot the load balancing value of our scheme when $M = 20, N = 100$, which is not the reward value of each training, but the average value of an episode. We can see that when the number of training steps exceeds 15,000,000, its average converges around -0.2. This means that after the model is trained, the sum of the load percentage differences between MEPs can be stably reduced to about 20% under the MEC application conditions given by us.

In Fig. 4(b), we compare the average delay of each MEC application providing services to users and the constrained delay when $M = 20$ and $N = 100$. We can see that the MEC application deployment decision output by the proposed algorithm can fully meet the delay requirements for users to obtain services. It should be pointed out that our randomly generated maximum tolerable delay l_j for each MEC application is relatively strict, so the algorithm decision cannot

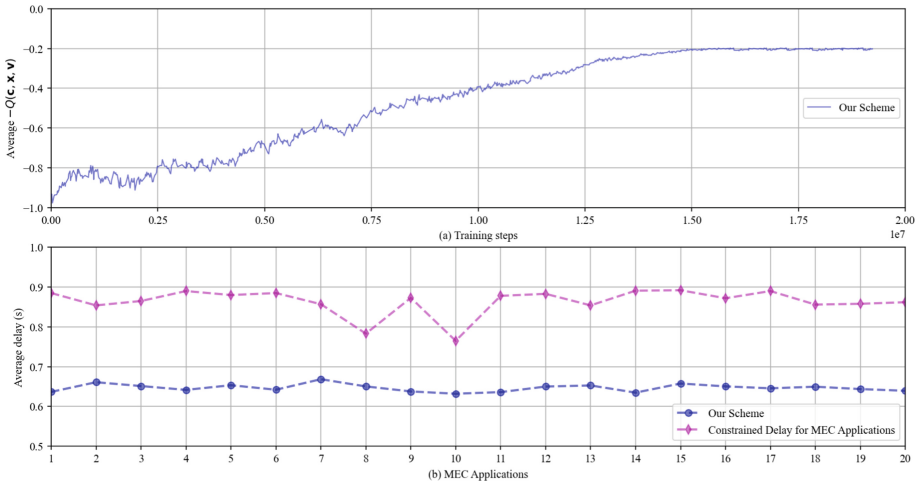


Fig. 4. Evaluated the convergence performance and service delay of our scheme when $M = 20$ and $N = 100$. (a) The load balancing value of our scheme. (b) Average delay for each MEC application to serve users, compared with constrained delay.

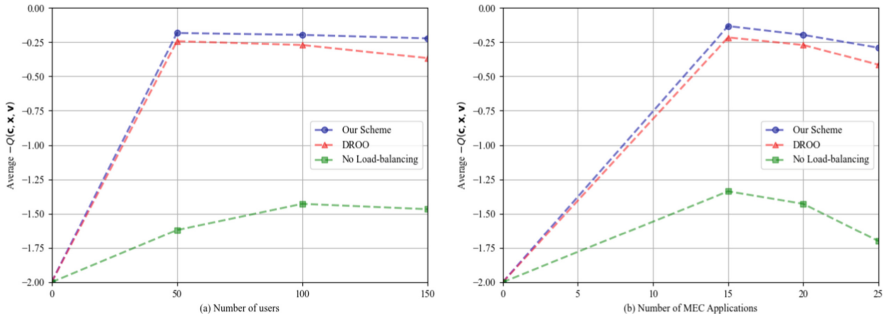


Fig. 5. (a) The load balancing effect of our scheme with different numbers of users when $M = 20$, and its comparison with the baseline. (b) The load balancing effect of our scheme with different numbers of MEC applications when $N = 100$, and its comparison with the baseline

be completely inclined to balance the load and ignore the delay constraint. At the same time, the computing resources required by each MEC application are also designed to be indivisible and of different sizes. Therefore, the load balance value in Fig. 4(a) can only be reduced to 20%, and then it cannot be further reduced due to the constraints of delay and computing resources.

In Fig. 5, we compare the load balancing capabilities of our scheme and the baseline under different numbers of users and MEC applications. In Fig. 5(a), we show the load balancing value when the number of users is 50, 100, and 150 when $M = 20$. As can be seen from the figure, both our scheme and the DROO

algorithm have a certain load balancing ability, and they are both significantly better than the unbalanced load algorithm. Further comparison of our scheme and DROO shows that our scheme is more stable when the number of users changes, while DROO has a relatively significant performance degradation when dealing with a larger state space. In Fig. 5(b), we show the load balancing value when the number of MEC applications is 15, 20, and 25 when $N = 100$. It can be seen from the figure that both our scheme and DROO are significantly better than the unbalanced load algorithm, but both show a certain degree of performance degradation when M increases. On the one hand, the increase of the number of applications will bring more strict delay constraints, which will lead to the decrease of the optimal value of the target. On the other hand, the increase of M will also bring larger action space, affecting the convergence performance of the algorithm. At the same time, it can also be seen that when M increases from 20 to 25, the performance loss of the value-based DROO algorithm is larger than that of our scheme, which reflects that the policy-based deep reinforcement learning is better than the value-based algorithm in the large action space.

5 Conclusion

In this paper, we propose an online MEC applications deployment algorithm based on deep reinforcement learning to balance the computational load among MEPs while ensuring that the services requested by users are satisfied. The algorithm learns from past deployment experience to improve the deployment actions generated by the actor network through reinforcement learning, and has a good convergence effect in a large action space. Compared with traditional optimization methods, the proposed algorithm does not need to solve NP-hard optimization problems at all. The simulation results show that, compared with the benchmark scheme, our scheme has the optimal load balancing effect under different number of users and MEC applications, and can satisfy dynamic user service requests at the same time. As future work, we will continue to study the performance of our scheme under burst application requests.

Acknowledgement. This work was supported in part by the National Key R&D Program of China under Grant 2019YFE0114000, in part by the Shenzhen science and technology innovation commission free exploring basic research project (No. 2021Szvup012), in part by the 111 Project of China (No B16006), and the research foundation of Ministry of Education China Mobile under Grant MCM20180101

References

1. Gupta, A., Jha, R.K.: A survey of 5G network: architecture and emerging technologies. IEEE Access **3**, 1206–1232 (2015)
2. Ksentini, A., Frangoudis, P.A., PC, A., Nikaein, N.: Providing low latency guarantees for slicing-ready 5G systems via two-level MAC scheduling. IEEE Netw. **32**(6), 116–123 (2018)

3. Taleb, T., Samdanis, K., Mada, B., Flinck, H., Dutta, S., Sabella, D.: On multi-access edge computing: a survey of the emerging 5G network edge cloud architecture and orchestration. IEEE Commun. Surv. Tutor. **19**(3), 1657–1681, thirdquarter (2017)
4. Mobile Edge Computing (MEC); Framework and Reference Architecture, ETSI Group Specification MEC 003 (2016)
5. Brik, B., Frangoudis, P.A., Ksentini, A.: Service-oriented MEC applications placement in a federated edge cloud architecture. In: ICC 2020–2020 IEEE International Conference on Communications (ICC), pp. 1–6 (2020)
6. Cziva, R., Anagnostopoulos, C., Pezaros, D.P.: Dynamic, latency-optimal VNF placement at the network edge. In: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, pp. 693–701 (2018)
7. Kiran, N., Liu, X., Wang, S., Yin, C.: Optimising resource allocation for virtual network functions in SDN/NFV-enabled MEC networks. IET Commun. **15**, 1710–1722 (2021)
8. Min, M., Xiao, L., Chen, Y., Cheng, P., Wu, D., Zhuang, W.: Learning-based computation offloading for IoT devices with energy harvesting. IEEE Trans. Veh. Technol. **68**(2), 1930–1941 (2019)
9. Chen, X., Zhang, H., Wu, C., Mao, S., Ji, Y., Bennis, M.: Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. IEEE Internet Things J. **6**(3), 4005–4018 (2019)
10. Li, J., Gao, H., Lv, T., Lu, Y.: Deep reinforcement learning based computation offloading and resource allocation for MEC. IEEE Wirel. Commun. Network. Conf. (WCNC) **2018**, 1–6 (2018)
11. Tang, M., Wong, V.W.S.: Deep reinforcement learning for task offloading in mobile edge computing systems. IEEE Trans. Mob. Comput. **21**(6), 1985–1997 (2022)
12. Huang, L., Bi, S., Zhang, Y.-J.A.: Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. IEEE Trans. Mob. Comput. **19**(11), 2581–2593 (2020)
13. Rhee, I., Shin, M., Hong, S., Lee, K., Kim, S.J., Chong, S.: On the levy-walk nature of human mobility. IEEE/ACM Trans. Network. **19**(3), 630–643 (2011)
14. 5G; Study on channel model for frequencies from 0.5 to 100 GHz, 3GPP TR 38.901 (2018)
15. Xu, C., Zhu, R., Yang, D.: Karting racing: a revisit to PPO and SAC algorithm. In: International Conference on Computer Information Science and Artificial Intelligence (CISAI) 2021, pp. 310–316 (2021)
16. Kim, S., Wimmer, H., Kim, J.: Analysis of deep learning libraries: Keras, PyTorch, and MXnet. In: 2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA), pp. 54–62 (2022)